

Assignment - 2

Indian Institute of Technology Tirupati

CS508 Distributed Systems

Logical Clocks and Message Ordering

Ordering Techniques Implemented

- **FIFO :-** A FIFO ordered protocol guarantees that messages by the same sender are delivered in the order that they were sent. That is, if a process multicasts a message m before it multicasts a message m' , then no correct process receives m' unless it has previously received m .
- **Casual :-** If message m_0 is causally dependent on message m , any correct process must deliver m before m_0 . Implemented heavily in social networks, new groups, comments on web etc., Causal ordering automatically implies FIFO ordering, but reverse is not true.
- **Total :-** If a correct process p delivers message m before m_0 (independent of the senders), then any other correct process that delivers m_0 would already have delivered m . Ensures all receivers receive all messages to the group in the same order. Doesn't care about the order in which they were sent by different senders. Used for consistent updates on replicated servers.

Implementation

Different methods used in code and there working :

- **Class OrderingSystemFifo**

The system maintains a buffer of messages that have been sent but not yet received, and ensures that messages are delivered in the order they were sent. The OrderingSystemFifo class has three instance variables:

numProcesses: the total number of processes in the distributed system.

buffer: a list of Message objects representing the messages that have been sent but not yet received.

localClocks: an array of integers representing the vector clocks for each process in the system.

The constructor takes in the numProcesses parameter and initializes the instance variables.

- The **sendMessage method** takes a Message object as a parameter, updates the local clock for the sending process, sets the vector clock for the message, adds a delay (specified by the delay field of the message), creates a new message object with the updated vector clock, and adds the new message to the buffer.
- The **isBufferEmpty method** simply returns whether the buffer is empty.
- The **receiveMessage method** retrieves the message with the lowest timestamp (i.e., the oldest message in the buffer), sleeps for the delay specified by the message, updates the local clock for the receiving process, removes the message from the buffer, and prints out a message indicating that the message has been received.

- **Class OrderingSystemCausal**

The OrderingSystemCausal class has three instance variables: numProcesses, which is the number of processes in the system; buffer, which is the list of messages that have been sent but not yet delivered; and localClocks, which is an array of integers representing the local clock for each process. The constructor initializes these variables.

- The **sendMessage method** sends a message by first updating the local clock for the sending process, then creating a vector clock for the message by cloning the current local clock. It then adds a delay to the message (to simulate network delays), creates a new message object with the vector clock and delay, and adds it to the buffer. Finally, it prints a message to the console indicating the sender, receiver, vector clock, and content of the message.
- The **isBufferEmpty method** simply returns a boolean indicating whether the buffer is empty or not.

- The **receiveMessage method** retrieves the message with the lowest timestamp (i.e., the oldest message) from the buffer, based on the vector clock of the sending process. It then adds a delay to the message (to simulate network delays), checks whether the message can be delivered based on the causal ordering constraints (i.e., whether the vector clock of the sending process is less than or equal to the local clock for each process), and if so, updates the local clock for the receiving process, removes the message from the buffer, and prints a message to the console indicating the receiver, vector clock, and content of the message. If the message cannot be delivered yet, the method simply prints a message to the console indicating that the message cannot be delivered yet.
- **Class OrderingSystemTotal**
The class has the following instance variables:

numProcesses : an integer representing the total number of processes involved in the system.

Buffer : a List object containing the messages in the buffer.

localClocks : an integer array representing the vector clocks of the local processes.

globalSequenceNumber : an integer representing the current global sequence number.

The class has a constructor that takes an integer representing the total number of processes as its argument and initializes the instance variables.

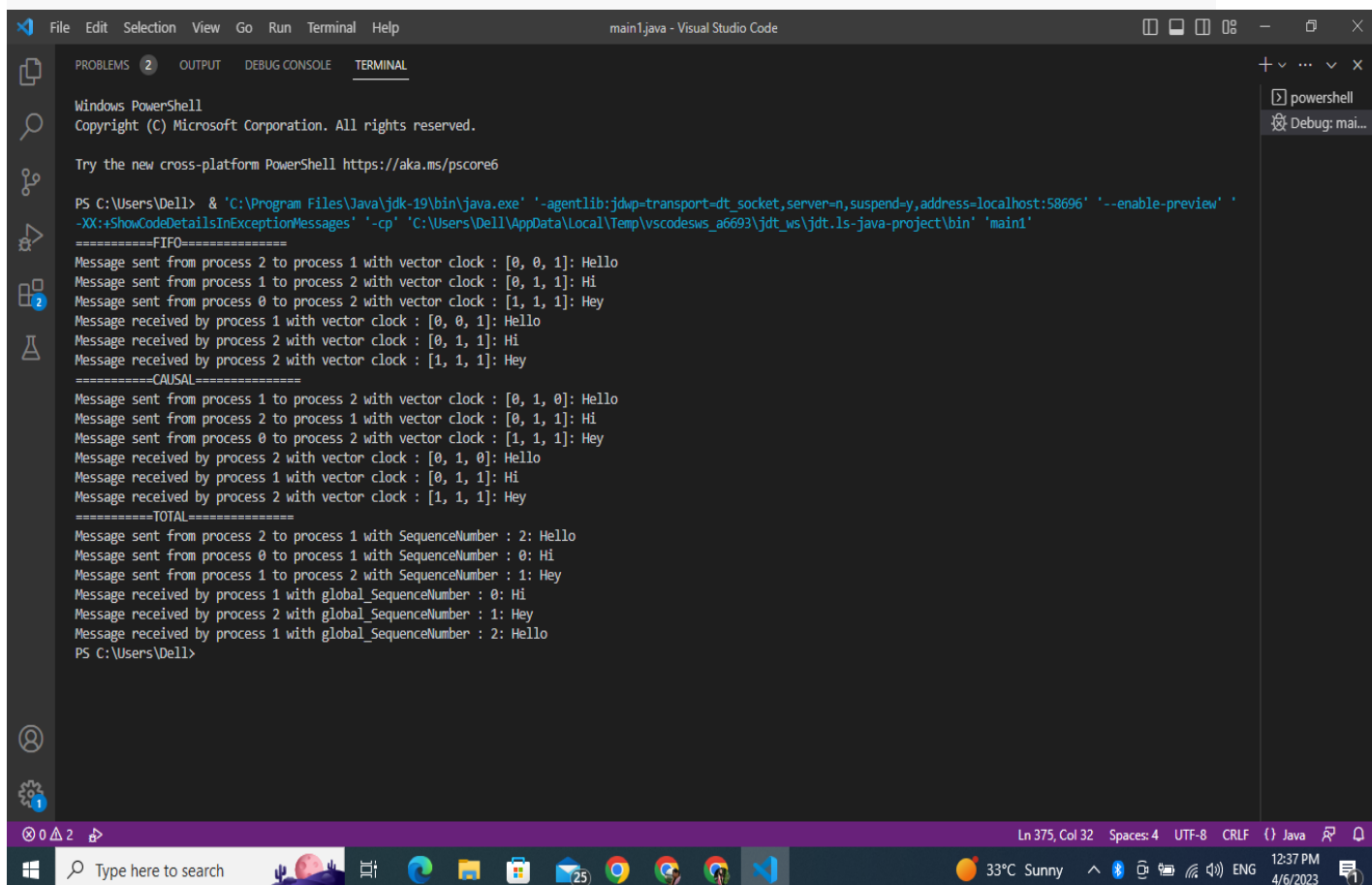
- **sendMessage Method**: a synchronized method that takes a "Message1" object as its argument and adds it to the buffer. It updates the local clock, adds a delay to the message, and increments the global sequence number. It also prints a message indicating that a message has been sent.
- **isBufferEmpty Method** : a synchronized method that returns a boolean indicating whether the buffer is empty.
- **receiveMessage**: a synchronized method that removes the message with the lowest global sequence number from the buffer, updates the local clock, and prints a message indicating that a message has been received.

The class uses the "Message1" class to represent messages, which contains the following instance variables:

- "senderId": an integer representing the ID of the sending process.
- "receiverId": an integer representing the ID of the receiving process.
- "content": a string representing the content of the message.
- "numProcesses": an integer representing the total number of processes involved in the system.
- "delay": an integer representing the delay of the message.
- "globalSequenceNumber": an integer representing the global sequence number of the message.
- "vectorClock": an integer array representing the vector clock of the message.

Demonstration

Output :



```
main1.java - Visual Studio Code
PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\Dell> & 'C:\Program Files\Java\jdk-19\bin\java.exe' '-agentlib:jdwp=transport=dt_socket,server=n,suspend=y,address=localhost:58696' '--enable-preview' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\Dell\AppData\Local\Temp\vscodesws_a6693\jdt_ws\jdt.ls-java-project\bin' 'main1'
=====FIFO=====
Message sent from process 2 to process 1 with vector clock : [0, 0, 1]: Hello
Message sent from process 1 to process 2 with vector clock : [0, 1, 1]: Hi
Message sent from process 0 to process 2 with vector clock : [1, 1, 1]: Hey
Message received by process 1 with vector clock : [0, 0, 1]: Hello
Message received by process 2 with vector clock : [0, 1, 1]: Hi
Message received by process 2 with vector clock : [1, 1, 1]: Hey
=====CAUSAL=====
Message sent from process 1 to process 2 with vector clock : [0, 1, 0]: Hello
Message sent from process 2 to process 1 with vector clock : [0, 1, 1]: Hi
Message sent from process 0 to process 2 with vector clock : [1, 1, 1]: Hey
Message received by process 2 with vector clock : [0, 1, 0]: Hello
Message received by process 1 with vector clock : [0, 1, 1]: Hi
Message received by process 2 with vector clock : [1, 1, 1]: Hey
=====TOTAL=====
Message sent from process 2 to process 1 with SequenceNumber : 2: Hello
Message sent from process 0 to process 1 with SequenceNumber : 0: Hi
Message sent from process 1 to process 2 with SequenceNumber : 1: Hey
Message received by process 1 with global_SequenceNumber : 0: Hi
Message received by process 2 with global_SequenceNumber : 1: Hey
Message received by process 1 with global_SequenceNumber : 2: Hello
PS C:\Users\Dell>
```